Formalizing Hardware-Software Contracts in Lean

THEOREM PROVER

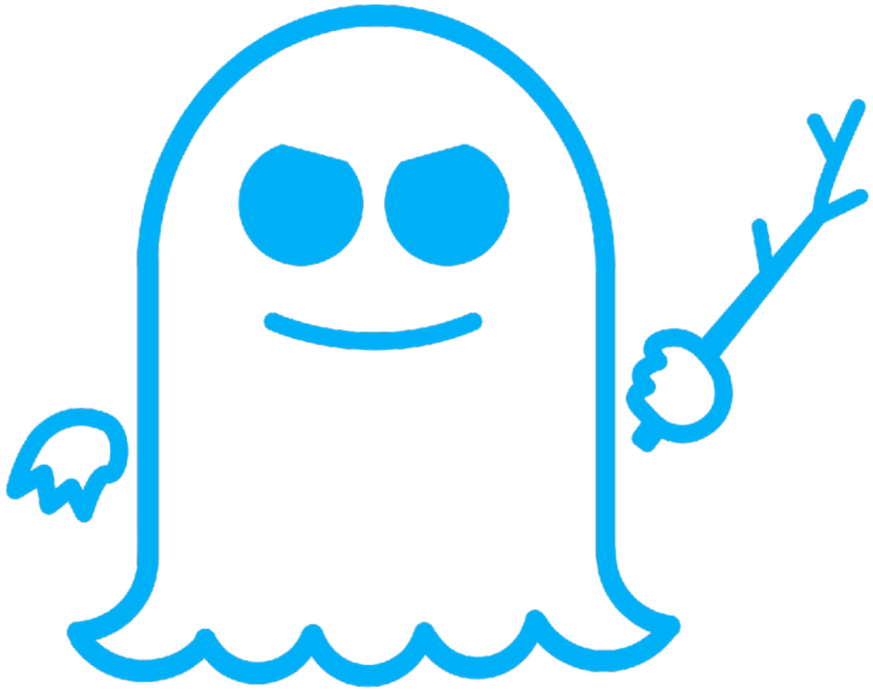
Haoyi Zeng

UNIVERSITÄT DES SAARLANDES

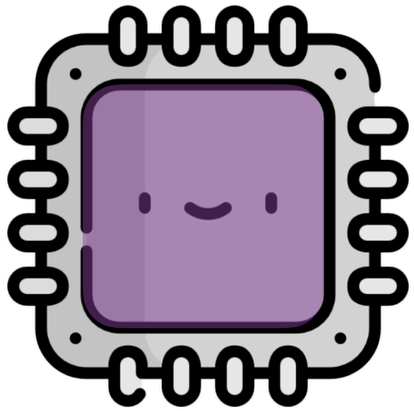
Thomas Bourgeat

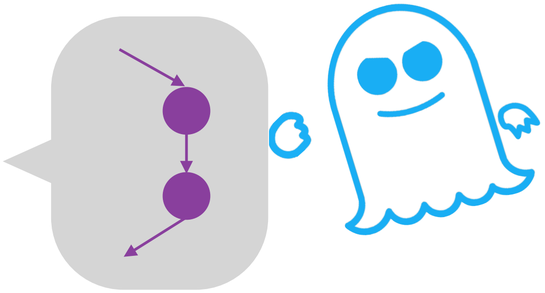
EPFL

29.11.2024



SPECTRE





Hardware

P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, Y. Yarom — | Spectre Attacks: Exploiting Speculative Execution — S&P 2019

Some Scary News

Researchers Disclose New Side-Channel Attacks Affecting All AMD CPUs

By Eduard Kovacs on October 15, 2021

RSS

s of new timing and power-based side-channel
MD, but the chipmaker says no new mitigations are

by researchers Moritz Lipp and Daniel Gruss of the
ael Schwarz of the CISPA Helmholtz Center for
those who discovered the original Meltdown and

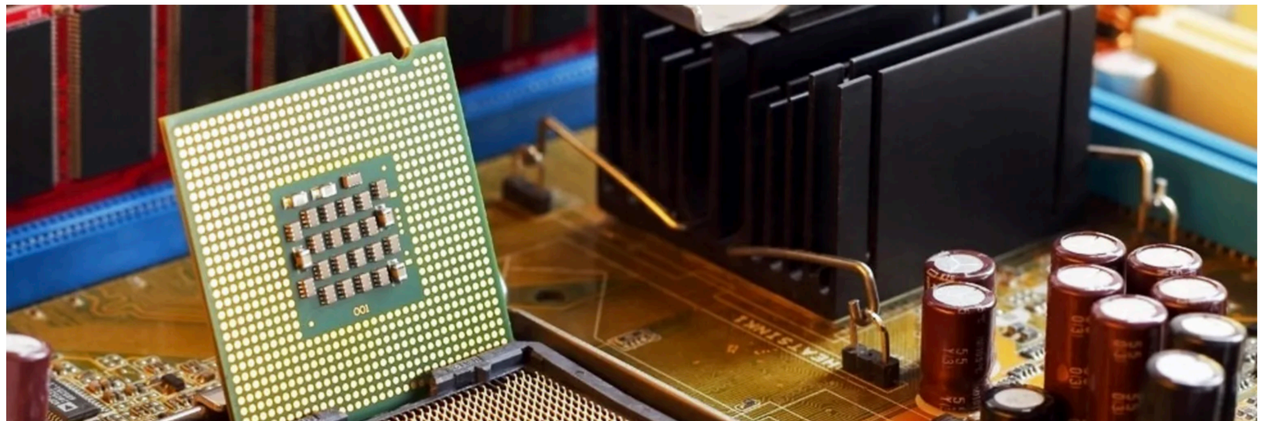
BBC

Home News Sport Business Innovation Culture Arts Travel Earth Video Live

Intel, ARM and AMD chip scare: What you need to know

4 January 2018

Share Save



Spectre Attacks

Timing side channels

+

Speculative execution

A Spectre Vulnerable program

Cannot access `A[x]` when `x` is out of bounds?

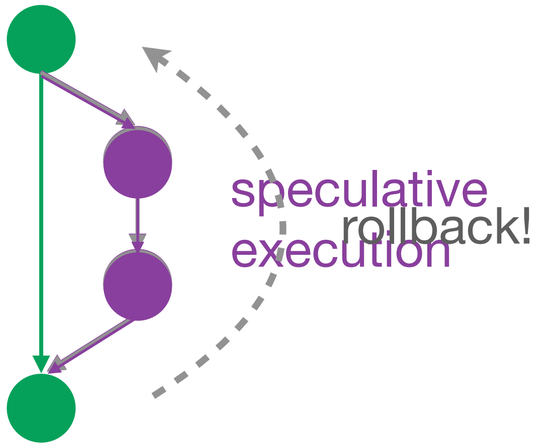
size of array `A`

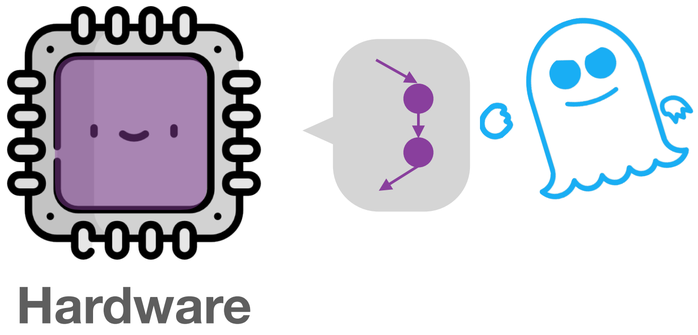
access “secret” `A[x]`

```
if (x < size){
  tmp = A[x]
  out = B[tmp]
}
```

leak `A[x]` via data cache

`(x < size) == true` ✗





How to formalize Spectre vulnerabilities?

How to formalize Spectre vulnerabilities?

Program π is **speculative non-interference (SNI)**: $\forall \sigma, \sigma'$: states,

$$[\![\pi]\!]^{seq}(\sigma) = [\![\pi]\!]^{seq}(\sigma') \implies [\![\pi]\!]^{spec}(\sigma) = [\![\pi]\!]^{spec}(\sigma')$$

Compare **leakage** without and with speculation

Speculative Execution

$$\textsc{Load}\quad \frac{p(a(\mathbf{pc})) = \mathbf{load}\ x, e \qquad x \neq \mathbf{pc} \qquad n = (\!|e|\!)(a)}{\langle m, a\rangle \to \langle m, a[\mathbf{pc} \mapsto a(\mathbf{pc}) + 1, x \mapsto m(n)]\rangle}$$

$[\![\pi]\!]^{seq}(\sigma)$ Sequence execution with events

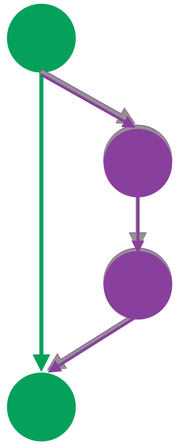
$$\textsc{Load}\quad \frac{p(a(\mathbf{pc})) = \mathbf{load}\ x, e \qquad \langle m, a\rangle \to \langle m', a'\rangle}{\langle m, a\rangle \xrightarrow[\text{ct}]{\texttt{load}\ (\!|e|\!)(a)}{}^{\text{seq}} \langle m', a'\rangle}$$

$[\![\pi]\!]^{spec}(\sigma)$ Speculative execution with events

Branch predictor

$$\textsc{Branch}\quad \frac{p(\sigma(\mathbf{pc})) = \mathbf{beqz}\ x, \ell \quad \ell_{correct} = \begin{cases} \ell & \text{if } \sigma(x) = 0 \\ \sigma(\mathbf{pc}) + 1 & \text{otherwise} \end{cases} \quad \ell_{mispred} \in \{\ell, \sigma(\mathbf{pc}) + 1\} \setminus \ell_{correct} \quad \omega_{mispred} = \begin{cases} \mathrm{w} & \text{if } \omega = \infty \\ \omega & \text{otherwise} \end{cases}}{\langle \sigma, \omega + 1\rangle \cdot s \xrightarrow[\text{ct}]{\texttt{pc}\ \ell_{mispred}}{}^{\text{spec}} \langle \sigma[\mathbf{pc} \mapsto \ell_{mispred}], \omega_{mispred}\rangle \cdot \langle \sigma[\mathbf{pc} \mapsto \ell_{correct}], \omega\rangle \cdot s}$$

```
if (x < size){
  tmp = A[x]
  out = B[tmp]
}
```



$$\sigma_1 = A + x \mapsto a$$

$$\sigma_2 = A + x \mapsto b$$

$$[\![\pi]\!]^{seq}(\sigma_1) = \text{start} \cdot \text{jump end}$$

$$[\![\pi]\!]^{seq}(\sigma_2) = \text{start} \cdot \text{jump end}$$

$$[\![\pi]\!]^{spec}(\sigma_1) = \text{start} \cdot \text{load } (A + x) \cdot \text{load } (a) \cdot \text{jump end}$$

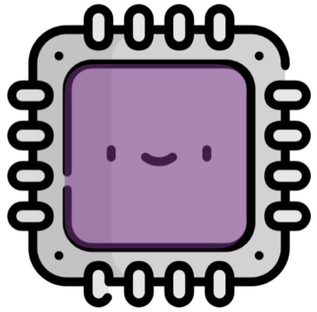
$$[\![\pi]\!]^{spec}(\sigma_2) = \text{start} \cdot \text{load } (A + x) \cdot \text{load } (b) \cdot \text{jump end}$$

Demo

Spectre-v1.lean

Speculative Execution

But that’s not true….



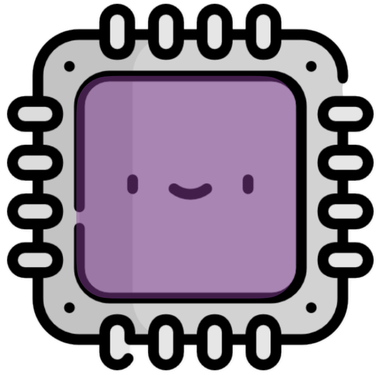
Hardware

There can be a huge gap between
the hardware and the model

$$[\![\cdot]\!]^{spec}$$

Develop an abstract model of hardware leakage

Key idea



Hardware

$\models$

$[\![\cdot]\!]$

Contract

$$[\![\pi]\!]^{seq}(\sigma) = [\![\pi]\!]^{seq}(\sigma') \implies [\![\pi]\!]^{spec}(\sigma) = [\![\pi]\!]^{spec}(\sigma')$$

$$[\![\pi]\!](\sigma) = [\![\pi]\!](\sigma') \implies (\!|\pi|\!)(\sigma) = (\!|\pi|\!)(\sigma')$$

Hardware Semantics

M. Guarnieri, B. Köpf, J. Reineke, and P. Vila
Hardware-Software Contracts for Secure Speculation
S&P (Oakland) 2021 (Best Paper Award 🏅)

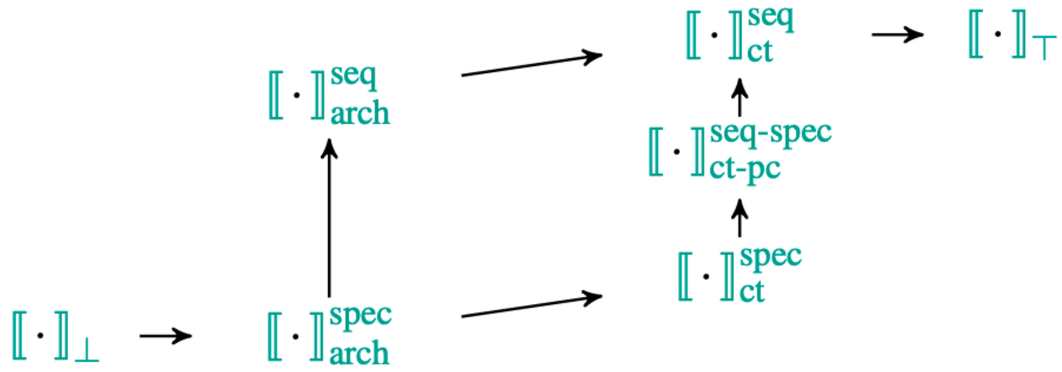
Hardware Software Contracts

$$[\![\bullet]\!] : \text{prog} \to \text{state} \to \text{List events}$$

$$[\![\bullet]\!] = \text{Execution Mode} \times \text{Observer Mode}$$

$$\text{Execution Mode} = \{\text{Seq}, \text{Spec}, \ldots\}$$

$$\text{Observer Mode} = \{\text{Arch}, \text{CT}, \ldots\}$$



Using Hardware-Software Contracts

From software side

$$[\![\pi]\!]^{seq}(\sigma) = [\![\pi]\!]^{seq}(\sigma') \implies [\![\pi]\!](\sigma) = [\![\pi]\!](\sigma')$$

```
if (x < size){
 tmp = A[x]
 out = B[tmp]
}
```

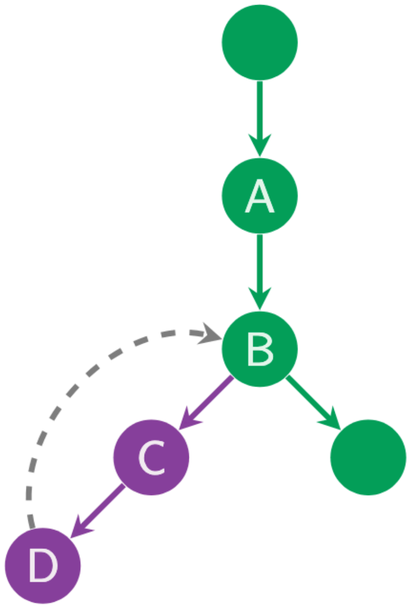
```
start:
  %cond ← %x < %size
  br %cond, then, end
then:
  load %tmp, %A + %x
  load %out, %B + %tmp
  br end
```

Spectector

Automatic detection of speculative information flows

Star 66

Spectre attacks



p

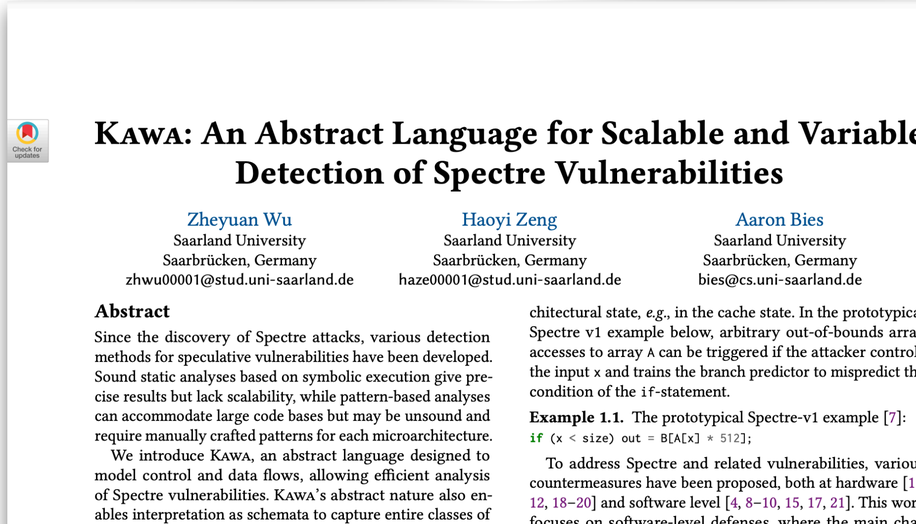
```
start:
  %cond ← ψ(%x, %size)
  br %cond, then, end

then:
  load %tmp, ψ(%A, %x)
  load %out, ψ(%B, %tmp)
  br end
```

φ_p

$\mathrm{flow}(\psi(\texttt{\%x}, \texttt{\%size}))$

- $\mathbf{start}(0)$
  - $\mathrm{block}(\mathit{then})$
  - $\mathrm{load}(\psi(\%A, \%x))$
  - $\mathbf{load}(\psi(\%B, \mathbf{read}(\mathbf{mem}, \psi(\%A, \%x))))$
  - $\mathrm{block}(\mathit{end})$
- $\mathbf{rollback}(0)$
- $\mathrm{block}(\mathit{end})$

Z3 $\neg\mathsf{Sat}(\varphi_p) \Rightarrow \mathsf{SNI}(p)$



Check for updates

KAWA: An Abstract Language for Scalable and Variable Detection of Spectre Vulnerabilities

Zheyuan Wu
Saarland University
Saarbrücken, Germany
zhwu00001@stud.uni-saarland.de

Haoyi Zeng
Saarland University
Saarbrücken, Germany
haze00001@stud.uni-saarland.de

Aaron Bies
Saarland University
Saarbrücken, Germany
bies@cs.uni-saarland.de

Abstract

Since the discovery of Spectre attacks, various detection methods for speculative vulnerabilities have been developed. Sound static analyses based on symbolic execution give precise results but lack scalability, while pattern-based analyses can accommodate large code bases but may be unsound and require manually crafted patterns for each microarchitecture.

We introduce KAWA, an abstract language designed to model control and data flows, allowing efficient analysis of Spectre vulnerabilities. KAWA's abstract nature also enables interpretation as schemata to capture entire classes of

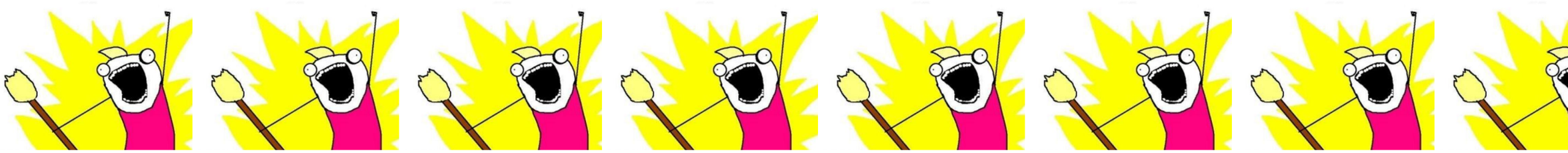
chitectural state, *e.g.*, in the cache state. In the prototypica Spectre v1 example below, arbitrary out-of-bounds arra accesses to array `A` can be triggered if the attacker control the input `x` and trains the branch predictor to mispredict th condition of the `if`-statement.

Example 1.1. The prototypical Spectre-v1 example [7]:

```
if (x < size) out = B[A[x] * 512];
```

To address Spectre and related vulnerabilities, variou countermeasures have been proposed, both at hardware [1 12, 18–20] and software level [4, 8–10, 15, 17, 21]. This wor focuses on software-level defenses, where the main cha

Let’s do everything using proof assistants



One more reason:

headConf(cr(*corr*$_{\text{cr,hr}}(i-1)(|buf_{i-1}|)$))[**pc** $\mapsto \ell$] because *headConf*(cr(*corr*$_{\text{cr,hr}}(i-1)(ln(\mathbf{hr}(i-1)))))(x) = 0$. From $\langle m_{i-1}, a_{i-1}\rangle \uplus buf_{i-1} = headConf(\text{cr}(corr_{\text{cr,hr}}(i-1)(|buf_{i-1}|)))$, we therefore get $headConf(rb_{\text{cr}}(corr_{\text{cr,hr}}(i-1)(ln(\mathbf{hr}(i-1)))) = \langle m_{i-1}, a_{i-1}\rangle \uplus buf_{i-1}[\mathbf{pc} \mapsto \ell]$. By leveraging $\cdot \uplus \cdot$'s definition and $\ell = \ell'$, we get $headConf(rb_{\text{cr}}(corr_{\text{cr,hr}}(i-1)(ln(\mathbf{hr}(i-1)))) = \langle m_{i-1}, a_{i-1}\rangle \uplus (buf_{i-1} \cdot \mathbf{pc} \leftarrow \ell' @ apl(a_{i-1}, buf_{i-1})(\mathbf{pc}))$. From $buf_i = \mathbf{pc} \leftarrow \ell' @ apl(a_{i-1}, buf_{i-1})(\mathbf{pc}) \cdot buf_{i-1}$, $a_i = a_{i-1}$, and $m_i = m_{i-1}$, we get $headConf(rb_{\text{cr}}(corr_{\text{cr,hr}}(i-1)(ln(\mathbf{hr}(i-1)))) = \langle m_i, a_i\rangle \uplus buf_i$. Finally, from $corr_{\text{cr,hr}}(i) = corr_{\text{cr,hr}}(i-1)[ln(\mathbf{hr}(i-1)) + 1 \mapsto rb_{\text{cr}}(corr_{\text{cr,hr}}(i-1)(ln(\mathbf{hr}(i-1))))]$, $|buf_i| = ln(\mathbf{hr}(i-1)) + 1$, $buf = buf_i$, we get $\langle m_i, a_i\rangle \uplus buf = headConf(\text{cr}(corr_{\text{cr,hr}}(i)(|buf|)))$.

(ii): We need to show $\#mispr(\langle m_i, a_i\rangle, buf) + 1 \geq |\text{cr}(corr_{\text{cr,hr}}(i)(|buf|))|$. From (H.3.a.ii) and $buf_{i-1} \in prefixes(buf_{i-1})$, we have $\#mispr(\langle m_{i-1}, a_{i-1}\rangle, buf_{i-1}) + 1 \geq |\text{cr}(corr_{\text{cr,hr}}(i-1)(|buf_{i-1}|))|$. From $buf = \mathbf{pc} \leftarrow \ell' @ apl(a_{i-1}, buf_{i-1})(\mathbf{pc}) \cdot buf_{i-1}$ and $headConf(\text{cr}(corr_{\text{cr,hr}}(i-1)(ln(\mathbf{hr}(i-1)))))(x) = 0$, we get that $\#mispr(\langle m_{i-1}, a_{i-1}\rangle, buf) = \#mispr(\langle m_{i-1}, a_{i-1}\rangle, buf_{i-1})$. From (2) and $p(apl(a_{i-1}, buf_{i-1})(\mathbf{pc})) = \mathbf{beqz}\ x, \ell$, we get that $\text{cr}(corr_{\text{cr,hr}}(i-1)(|buf_{i-1}|)+1)$ is obtained by executing the BRANCH rule of $\rightsquigarrow^{\text{spec}}_{\text{ct}}$ starting from $\text{cr}(corr_{\text{cr,hr}}(i-1)(|buf_{i-1}|))$. From this and Lemma 2, $|rb_{\text{cr}}(corr_{\text{cr,hr}}(i-1)(ln(\mathbf{hr}(i-1))))| = |\text{cr}(corr_{\text{cr,hr}}(i-1)(|buf_{i-1}|))|$. Therefore, we get $\#mispr(\langle m_{i-1}, a_{i-1}\rangle, buf) + 1 \geq |rb_{\text{cr}}(corr_{\text{cr,hr}}(i-1)(ln(\mathbf{hr}(i-1))))|$. Finally, from $a_i = a_{i-1}$, $m_i = m_{i-1}$, $corr_{\text{cr,hr}}(i) = corr_{\text{cr,hr}}(i-1)[ln(\mathbf{hr}(i-1)) + 1 \mapsto rb_{\text{cr}}(corr_{\text{cr,hr}}(i-1)(ln(\mathbf{hr}(i-1))))]$, and $|buf| = ln(\mathbf{hr}(i-1)) + 1$, we get $\#mispr(\langle m_i, a_i\rangle, buf) + 1 \geq |\text{cr}(corr_{\text{cr,hr}}(i)(|buf|))|$.

(iii): We need to show $\#mispr(\langle m_i, a_i\rangle, buf) = 0 \leftrightarrow headWndw(\text{cr}(corr_{\text{cr,hr}}(i)(|buf|))) = \infty$. From (H.3.a.iii) and $buf_{i-1} \in prefixes(buf_{i-1})$, we have $\#mispr(\langle m_{i-1}, a_{i-1}\rangle, buf_{i-1}) = 0 \leftrightarrow headWndw(\text{cr}(corr_{\text{cr,hr}}(i-1)(|buf_{i-1}|))) = \infty$. From (2) and $p(apl(a_{i-1}, buf_{i-1})(\mathbf{pc})) = \mathbf{beqz}\ x, \ell$, we get that $\text{cr}(corr_{\text{cr,hr}}(i-1)(|buf_{i-1}|) + 1)$ is obtained by executing the BRANCH rule

~ 80 pages

Using Hardware-Software Contracts

From hardware side

$$[\![\pi]\!](\sigma) = [\![\pi]\!](\sigma') \implies (\!|\pi|\!)(\sigma) = (\!|\pi|\!)(\sigma')$$

Given any

Component	States	Initial state	Functions	
Cache	*CacheStates*	cs_0	$access : CacheStates \times Vals \to \{\texttt{Hit},\texttt{Miss}\}$	$update : CacheStates \times Vals \to CacheStates$
Branch predictor	*BpStates*	bp_0	$predict : BpStates \times Vals \to Vals$	$update : BpStates \times Vals \times Vals \to BpStates$
Pipeline scheduler	*ScStates*	sc_0	$next : ScStates \to Dir$	$update : ScStates \times Bufs \to ScStates$

$(\!|\cdot|\!)$: prog → hardware_state → List events

→ retire

→ execute

→ fetch

EXECUTE-BRANCH-COMMIT

$$\frac{\ell_0 \neq \varepsilon \qquad p(\ell_0) = \textbf{beqz}\ x, \ell'' \qquad |buf| = i-1 \qquad a' = apl(buf, a) \qquad (a'(x) = 0 \wedge \ell = \ell'') \vee (a'(x) \in Vals \setminus \{0, \bot\} \wedge \ell = \ell_0 + 1) \qquad \textbf{spbarr@}T' \notin buf \qquad bp' = update(bp, \ell_0, \ell)}{\langle m, a, buf \cdot \textbf{pc} \leftarrow \ell@\ell_0 \cdot buf', cs, bp\rangle \xRightarrow{\textbf{execute}\ i} \langle m, a, buf \cdot \textbf{pc} \leftarrow \ell@\varepsilon \cdot buf', cs, bp'\rangle}$$

EXECUTE-BRANCH-ROLLBACK

$$\frac{|buf| = i-1 \qquad a' = apl(buf, a) \qquad \textbf{spbarr@}T' \notin buf \qquad \ell_0 \neq \varepsilon \qquad p(\ell_0) = \textbf{beqz}\ x, \ell'' \qquad (a'(x) = 0 \wedge \ell \neq \ell'') \vee (a'(x) \in Vals \setminus \{0, \bot\} \wedge \ell \neq \ell_0 + 1) \qquad \ell' \in \{\ell'', \ell_0 + 1\} \setminus \{\ell\} \qquad bp' = update(bp, \ell_0, \ell')}{\langle m, a, buf \cdot \textbf{pc} \leftarrow \ell@\ell_0 \cdot buf', cs, bp\rangle \xRightarrow{\textbf{execute}\ i} \langle m, a, buf \cdot \textbf{pc} \leftarrow \ell'@\varepsilon, cs, bp'\rangle}$$

Definition (Out-of-order scheduler)

S_{ooo} := fetch·fetch·execute·fetch·execute·retire···

Definition (Sequence scheduler)

S_{seq} := fetch·execute·retire·fetch·execute·retire···

Formalizing Hardware-Software Contracts

Goal

Theorem 1 (Contract Satisfaction 1):
For any hardware model $(\!|\pi|\!)$ instantiated by arbitrary cache, branch predictor, and scheduler, we have:

$$(\!|\cdot|\!) \models [\![\cdot]\!]^{spec}$$

$$[\![\pi]\!]^{seq}(\sigma) = [\![\pi]\!]^{seq}(\sigma') \implies [\![\pi]\!](\sigma) = [\![\pi]\!](\sigma')$$

$$[\![\pi]\!](\sigma) = [\![\pi]\!](\sigma') \implies (\!|\pi|\!)(\sigma) = (\!|\pi|\!)(\sigma')$$

Conclusion: For any program π, if Spectector/Kawa shows that π is SNI, then π is secure against side-channel attacks on this machine model

Fact 1 (Contract Satisfaction 1):
For any hardware model $(\!|\cdot|\!)$ instantiated by arbitrary cache, branch predictor. If the scheduler is S_{seq} , we have:

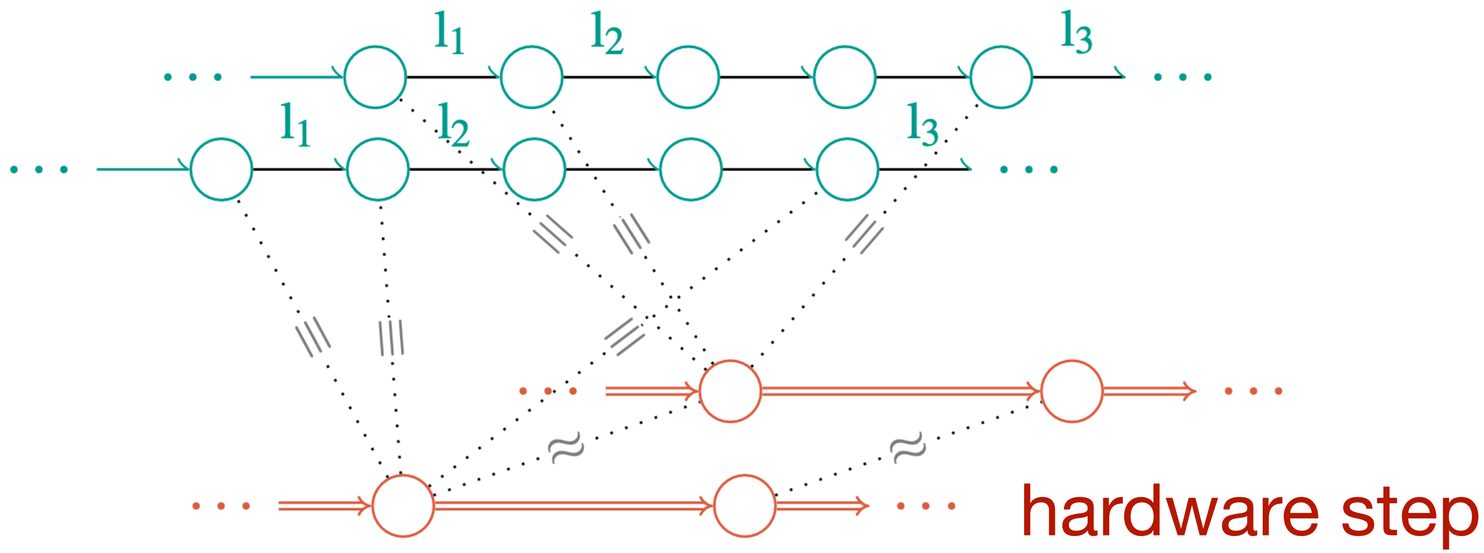
Because the scheduler is boring

$$(\!|\cdot|\!) \models [\![\cdot]\!]^{seq}$$

Proof by induction

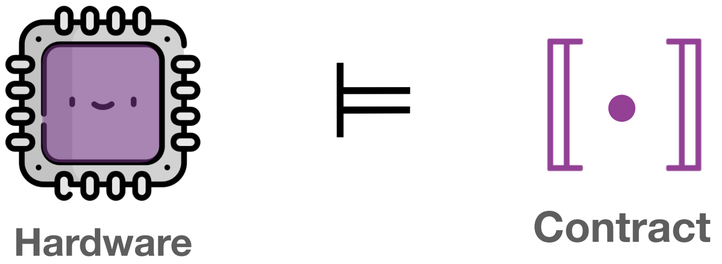
on contracts step

with an invariant



seqsat.lean

Future Work



1. **Challenge of contract satisfaction proof**
2. **A program logic for proving SNI**

   (Relational Hoare Logic, Hyper Hoare Logic)

Full stack verification

1. **More realistic examples** (Model Checking + Proof Assistant)
2. **Secure compilation**